

76582

80387 80-BIT CHMOS III NUMERIC PROCESSOR EXTENSION

- High Performance 80-Bit Internal Architecture
- Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic
- Five to Nine Times 8087/80287 Performance
- Expands 386™ CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands
- Directly Extends 386™ CPU Instruction Set to Include Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Data Types
- Upward Object-Code Compatible from 8087 and 80287
- Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT and LOGARITHM
- Built-In Exception Handling
- Operates Independently of Real, Protected and Virtual-8086 Modes of the 386™ Microprocessor
- Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack
- Available in 68-Pin PGA Package
(See Packaging Spec: Order #231369)

The Intel 80387 is a high-performance numerics processor extension that extends the Intel 386™ Architecture with floating point, extended integer and BCD data types. The 386™ Microprocessor and the 80387 Coprocessor computing system fully conforms to the ANSI/IEEE floating-point standard. Using a numerics oriented architecture, the 80387 adds over seventy mnemonics to the 386 Microprocessor instruction set, making the 386 Microprocessor and the 80387 Coprocessor a complete solution for high-performance numerics processing. The 80387 is implemented with 1.5 micron, high-speed CHMOS III technology and packaged in a 68-pin object-code compatible from the 386 Microprocessor and the 80287 Coprocessor, 80286/80287 and 8086/8087 computing systems.

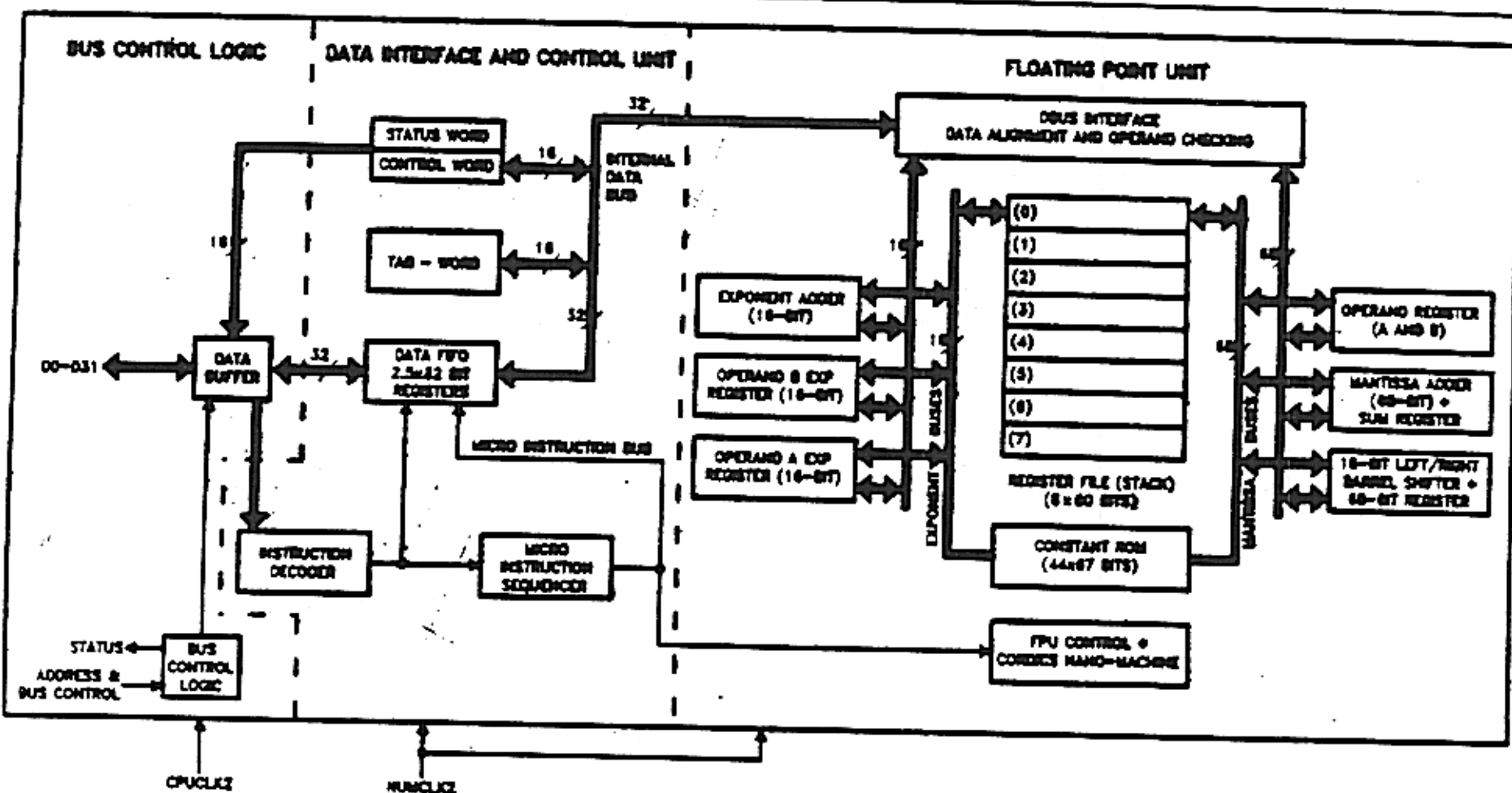


Figure 0.1. 80387 Block Diagram

231920-1

2.0 PROGRAMMING INTERFACE

The 80387 adds to the 386 Microprocessor system additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the 80387 requires no special programming tools, because all new instructions and data types are directly supported by the 386 CPU assembler and compilers for high-level languages. All 8086/8088 development tools that support the 8087 can also be used to develop software for the 386 Microprocessor and 80387 Coprocessor in real-address mode or virtual-8086 mode. All 80286 development tools that support the 80287 can also be used to develop software for the 386 Microprocessor and 80387 Coprocessor.

All communication between the 386 Microprocessor and the 80387 is transparent to applications software. The CPU automatically controls the 80387 whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the 80387. All memory addressing modes, including use of displacement, base register, index register, and scaling, are available for addressing numerics operands.

Section 6 at the end of this data sheet lists by class the instructions that the 80387 adds to the instruction set of the 386 Microprocessor system.

2.1 Data Types

Table 2.1 lists the seven data types that the 80387 supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses evenly divisible by four (doubleword boundaries); operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

Internally, the 80387 holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating-point numbers, or 18-digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

2.2 Numeric Operands

A typical NPX instruction accepts one or two operands and produces a single result. In two-operand instructions, one operand is the contents of an NPX register, while the other may be a memory location. The operands of some instructions are predefined; for example FSQRT always takes the square root of the number in the top stack element.

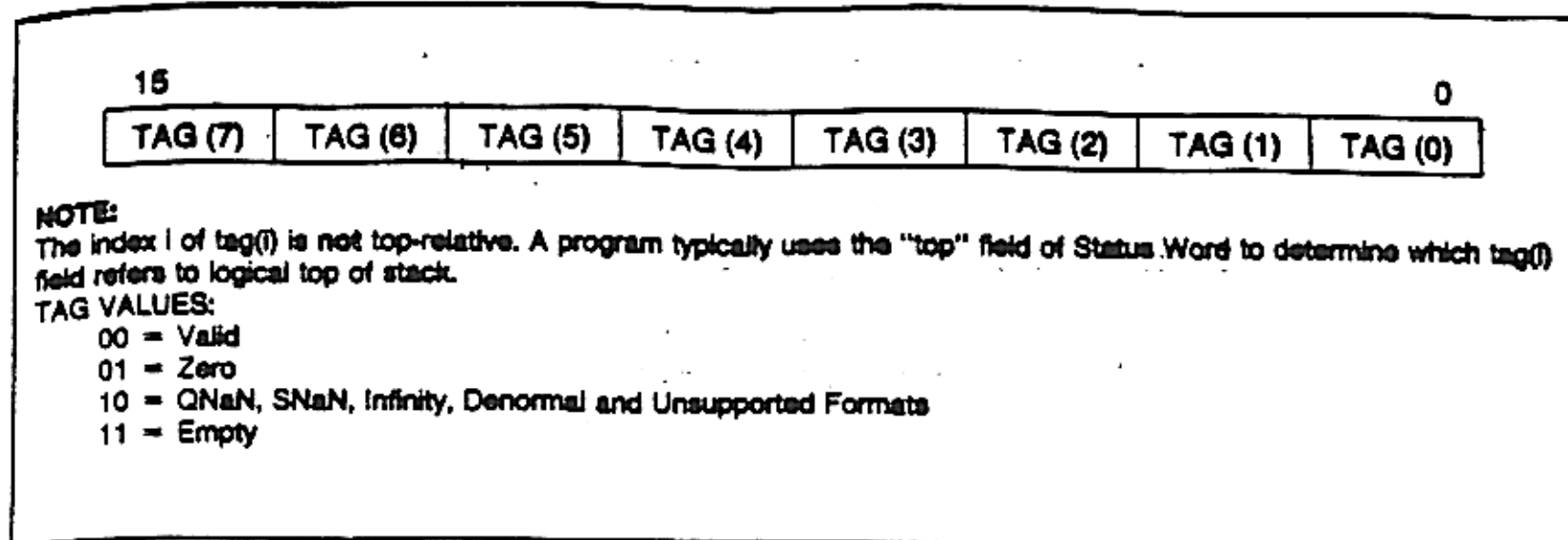


Figure 2.1. 80387 Tag Word

2.3 Register Set

Figure 1.1 shows the 80387 register set. When an 80387 is present in a system, programmers may use these registers in addition to the registers normally available on the 386 CPU.

2.3.1 DATA REGISTERS

80387 computations use the 80387's data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers in the 80387 is 80 bits wide and is divided into "fields" corresponding to the NPXs extended-precision real data type.

The 80387 register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments

TOP by one. Like the 386 Microprocessor stacks in memory, the 80387 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.3.2 TAG WORD

The tag word marks the content of each numeric data register, as Figure 2.1 shows. Each two-bit tag represents one of the eight numeric registers. The principal function of the tag word is to optimize the NPXs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

Table 2-2: Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2 Q0 Q1 or O/U#			Reduction— 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.4)		Zero or O/U#	Operand is not comparable (Table 2.4)
FXAM	Operand class (see Table 2.5)		Sign or O/U#	Operand class (Table 2.5)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FEXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1.	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			
O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).			
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.			
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.			
UNDEFINED	Do not rely on finding any specific value in these bits.			

2.3.4 INSTRUCTION AND DATA POINTERS

Because the NPX operates in parallel with the CPU, any errors detected by the NPX may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numeric instruction, the 386 Microprocessor and 80387 Coprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written error handlers. These registers are actually located in the 386 CPU, but appear to be located in the 80387 because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR. (In the 8086/8087 and 80286/80287, these registers are located in the NPX.) Whenever

the 386 CPU decodes a new ESC instruction, it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present), and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the 386 Microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the 386 Microprocessor is in virtual-8086 mode, the real-address mode formats are used. (See Figures 2.3 through 2.6.) The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the 386 Microprocessor registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

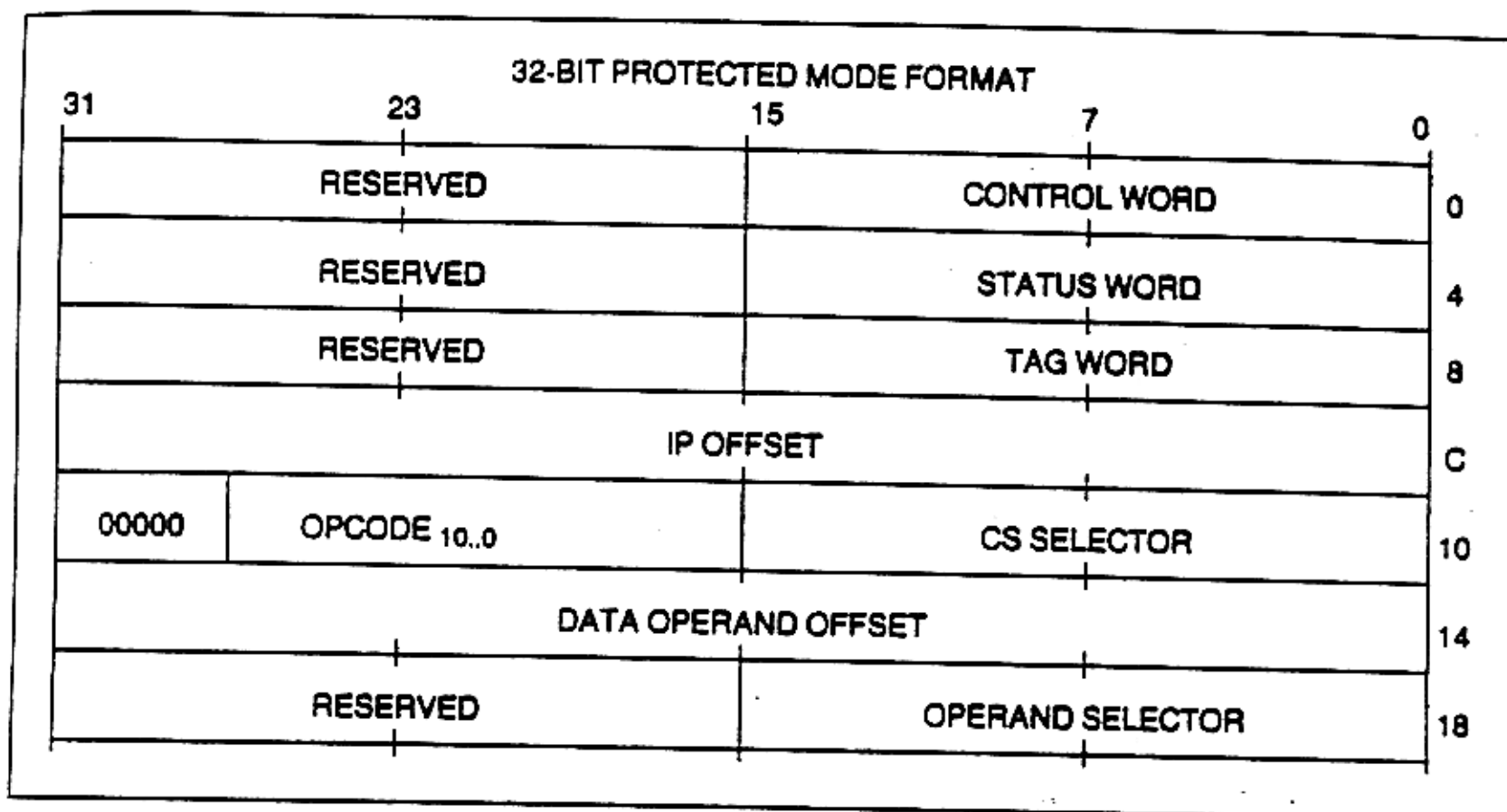


Figure 2.3. Protected Mode 80387 Instruction and Data Pointer Image in Memory, 32-Bit Format